2016级算法第一次上机解题报告

16041049 陈天宇

# 一、总结

本次上机主要考察知识点为分治、递推、简单数论等算法知识。

关于题目难度，我只能说，对于我这种只学过C语言的高工渣渣来说，每道题都很有挑战性。所以只能说对各题有个最粗浅的理解，其中A、C题主要考察了简单的递推，B题考察了数论，D题是对题目中公式的简单应用，E题是快排的变体，F题是归并排序求逆序数的变体，EF主要应用了分治、递归的思想。个人认为，递推这种方法，主要是要通过一些简单的情况快速发现规律，这个需要大量题目的训练，我在第一题错排和第三题递推上耗费了大量时间寻找递推公式，最后导致其他题目没有充足的时间思考。还有在使用数组的时候能long long尽量不要int，D题就因为数据的问题一直wa。所以，对于递推的问题，一定要多加练习，加快解题的速度，还有在数组的使用中一定要注意数据格式。

针对这次上机暴露出的和软院专业知识的差距，在接下来两周中，我将着重自学数据结构中的递推，树，数论的相关内容，并开始大量刷oj的递推题。近期可能会多多劳烦助教帮助了。

# 二、解题报告

### A the stupid owls

### 思路分析

本题是一道典型的递推问题。

这道题翻译成简单的问题：n只猫头鹰给n个人送信，求每个人收到的都是错信的概率。

不妨设n只猫头鹰送错信的情况总数为D（n），而所有的情况总数为S(n) = n!，概率为D（n）/S(n)

那么问题就转换为求n个数错排的情况总数D（n），我们需要通过分析来推导出D（n）的递推公式为

D（n）= （n-1）[D(n-2) + D(n-1)]

但是，由于出现了阶乘，在n>12时，结果将远远超出long long的范围（显然测试点小于13，这应该是一个bug），所以考虑用数学分析的方法求这个数列的极限，在n>7时，概率将趋向于一个极限，而这个极限的精度远高于两位小数，所以可以近似地视为36.79%。

### 算法分析

1. D（n）公式的推导

第一步，把第n个元素放在一个位置，比如位置k，一共有n-1种方法；

第二步，放编号为k的元素，这时有两种情况：⑴把它放到位置n，那么，对于剩下的n-1个元素，由于第k个元素放到了位置n，剩下n-2个元素就有D(n-2)种方法；⑵第k个元素不把它放到位置n，这时，对于这n-1个元素，有D(n-1)种方法；

综上得到

D(n) = (n-1) [D(n-2) + D(n-1)]

特殊地，D(1) = 0, D(2) = 1.

1. 概率极限存在的证明

下面推导通项公式：

设D(k) = k! N(k), k = 1, 2, …, n,

则N(1) = 0, N(2) = 1/2.

n ≥ 3时，n! N(n) = (n-1) (n-1)! N(n-1) + (n-1)! N(n-2)

即 nN(n) = (n-1) N(n-1) + N(n-2)

于是有N(n) - N(n-1) = - [N(n-1) - N(n-2)] / n = (-1/n) [-1/(n-1)] [-1/(n-2)]…(-1/3) [N(2) - N(1)] = (-1)^n / n!.

因此

N(n-1) - N(n-2) = (-1)^(n-1) / (n-1)!,

N(2) - N(1) = (-1)^2 / 2!.

相加，可得

N(n) = (-1)^2/2! + … + (-1)^(n-1) / (n-1)! + (-1)^n/n!

因此

D(n) = n! [(-1)^2/2! + … + (-1)^(n-1)/(n-1)! + (-1)^n/n!].

接着证明极限存在，设概率为P

P (n)= D(n)/n! = [(-1)^2/2! + … + (-1)^(n-1)/(n-1)! + (-1)^n/n!]

P(n) – P(n-1) = (-1)^n/n!

lim(n->∞) (P(n) – P(n-1)) = 0

而当n>7时，|P(n) – P(n-1)|<0.00001

即P约等于P（7） = 36.79%

注：参考代码一是AC代码，但未考虑阶乘的溢出，参考代码二是针对这道题的精度给出的正确代码。

参考资料：百度百科-错排公式

。

### 参考代码一

1. #include<iostream>
2. #include<iomanip>
3. using namespace std;
4. long jiecheng(int n)
5. {
6. long ans =1;
7. for(int i=1;i<=n;i++)
8. {
9. ans \*= i;
10. }
11. return ans;
12. }
13. int main()
14. {
15. long arr[26];
16. arr[1] = 0;
17. arr[2] = 1;
18. for(int i=3;i<=25;i++)
19. {
20. arr[i] = (i-1)\*(arr[i-2]+arr[i-1]);
21. }
22. int n;
23. long sum;
24. double ans = 0.0;
25. while(cin>>n)
26. {
27. sum = jiecheng(n);
28. ans = (arr[n]\*100.00/sum);
29. cout<<fixed<<setprecision(2)<<ans<<"%"<<endl;
30. }
31. }

### 参考代码二

1. #include<iostream>
2. #include<iomanip>
3. using namespace std;
4. long long jiecheng(int n)
5. {
6. long ans =1;
7. for(int i=1;i<=n;i++)
8. {
9. ans \*= i;
10. }
11. return ans;
12. }
13. int main()
14. {
15. long arr[26];
16. arr[1] = 0;
17. arr[2] = 1;
18. for(int i=3;i<=25;i++)
19. {
20. arr[i] = (i-1)\*(arr[i-2]+arr[i-1]);
21. }
22. int n;
23. long long sum;
24. double ans = 0.0;
25. while(cin>>n)
26. {
27. if(n<8)
28. {
29. sum = jiecheng(n);
30. ans = (arr[n]\*100.00/sum);
31. }
32. else
33. {
34. ans = 36.79;
35. }
36. cout<<fixed<<setprecision(2)<<ans<<"%"<<endl;
37. }
38. }

### B  ModricWang和数论

### 思路分析

本题是一道纯粹的数论问题。

对于任意一个正自然数n，显然它除1以外的最小约数一定大于等于2，即说明对于任何x（n/2<x<=n），n%x必定不同，此外当x>n时，n%x = n。注意奇偶性的分类即可。

### 算法分析：

当n为奇数时，有（n+1）/2个数大于n/2,再加上n本身，结果为（n+3）/2，

当n为偶数时，有n/2个数大于n/2，再加上n本身，结果为（n+2）/2。

### 参考代码

1. #include<iostream>
2. using namespace std;
3. int main()
4. {
5. long long a;
6. while(cin>>a)
7. {
8. long long ans;
9. if(a == 1)
10. {
11. ans = 2;
12. }
13. else if( a%2 == 0 )
14. {
15. ans = a/2 + 1;
16. }
17. else
18. {
19. ans = (a + 1)/2 + 1;
20. }
21. cout<<ans<<endl;
22. }
23. }

### C AlvinZH去图书馆

### 思路分析

本题典型动态规划题。

这道题与平常动归的区别在于达到n步数有两条路径，一条是n-1步数的方法加上n-2步数的方法，另一条路径是等于n-3步数的方法，但是第二条路径不能连续两次使用。

于是我一开始想的状态转移方程是这样的arr[i] = arr[i-1] + arr[i-2] + arr[i-3] – arr[i-6]，但是这个方程有一个非常明显的问题，那就是两次计算后就会多减去一部分arr[i-6]，可是在arr[i+3]里实际上是存在一个arr[i-6]的，所以对于这两条不同的路径，最好开两个数组进行计算。

正确的转移方程应该是a[i] = a[i-1] + a[i-2] +b[i-1] + b[i-2]，a[i]表示第i步的方法数是i-1步和i-2步方法之和，而b[i]表示当第i步的方法数是第i-3步的方法数，即b[i] = a[i-3]，第i步的方法数为两条路径结果之和，也就是a[i]+b[i]。

### 算法分析

复杂度O（n）。

注：参考资料：http://blog.csdn.net/Winjay\_233/article/details/78244599

### 参考代码

1. #include<cstdio>
2. #define maxn 57
3. long long a[maxn],b[maxn];
4. void init()
5. {
6. a[1] = 1;
7. a[2] = 2;
8. a[3] = 3;
9. b[1] = 0;
10. b[2] = 0;
11. b[3] = 1;
12. }
13. int main()
14. {
15. int n;
16. init();
17. for(int i = 4;i <= 50;i++)
18. {
19. a[i] = a[i-1] + a[i-2] + b[i-1] + b[i-2];
20. b[i] = a[i-3];
21. }
22. while(~scanf("%d",&n))
23. {
24. printf("%lld\n",a[n]+b[n]);
25. }
26. }

### D 水水的horner rule

### 思路分析

这个题就是利用horner rule实现O（n）的进制转换，由H进制数a转十进制数y的公式

y = a0 + a1\*H + a2\*H^2 + a3\*H^3 + …… + an\*H^n

需要注意的坑点是数据类型，题目x的十进制表示其实是会超过int的，所以还是能用long long尽量不要用int。然后x最好用字符串表示，再计算是每位的数字应减去’a’的ASCII码。

### 算法分析

算法复杂度为O(n)，一个for循环over。

### 参考代码一

1. #include<iostream>
2. #include<string.h>
3. #include<string>
4. using namespace std;
5. long long horner\_rule(int H,string x)
6. {
7. int y =0;
8. for(int i=0;i<x.size();i++)
9. {
10. y = x[i] - '0' + H\*y;
11. }
12. return y;
13. }
14. int main()
15. {
16. int n;
17. cin>>n;
18. for(int i=0;i<n;i++)
19. {
20. int H1,H2;
21. string x1,x2;
22. cin>>H1>>x1;
23. cin>>H2>>x2;
24. long long sum = horner\_rule(H1,x1) + horner\_rule(H2,x2);
25. cout<<sum<<endl;
26. }
27. }

### E ModricWang’s QuickSort

### 思路分析

本题是快排的一种划分方式，按照题目中的算法

1.设数组为arr[n]，元素从0开始存储

2.令i=0，j=n-1, mid=arr[n/2]  
3.如果 i < j，转到4，否则转到7  
4.如果arr[i] <= mid, i++ ，重复执行直到arr[i]>mid  
5.如果arr[j] > mid, j— ，重复执行直到arr[j]<=mid  
6.如果i < j, 交换arr[i]和arr[j]，转到4  
7.退出

需要注意的是每次递归后的中心轴元素的位置是会发生变化的，所以要用key储存它的值，另一方面题目的划分不是按照中间元素最后在哪里划分而是将下标作为中间轴划分，最后i=j的位置才是所谓的中间轴。

题目的要求是输出两次划分后的第二部分，所以与其写递归，不如直接调用两次函数后直接输出值来的方便。所以我在第一次划分后记录下中间轴的位置b，然后第二次只针对第一次划分的左边进行划分，将中间轴的位置记为c，输出c到b中所有的元素。

### 算法分析

快排算法的复杂度是O(nlgn)，但是快排是一种不稳定的算法，如果给的数据集较小，或者数据集中含有大量相同元素，按照题目中的划分方法，复杂度将变为O(n^2)。

因此我们可以对该快排算法进行进一步优化，在选取key进行比较划分的时候，我们不妨选取数据的开始元素，结束元素和中间元素的中间值，这样的话，如果数据是逆序或者顺序的，划分会比较平均，如果数据是正态分布，那么也可以有效地把数据均匀分开。在数据元素较小时可采用冒泡排序。

注：参考代码一是AC代码，参考代码二是题目中快排算法的优化。

### 参考代码一

1. #include<bits/stdc++.h>
2. using namespace std;
3. const int N = 1000001;
4. int arr[1000001];
5. int tmp;
6. int main()
7. {
8. int n;
9. cin>>n;
10. for(int i=0;i<n;i++)
11. {
12. cin>>arr[i];
13. }
14. int i=0,j=n-1;
15. int mid = arr[n/2];
16. while(i<j)
17. {
18. while(arr[i]<=mid && i<=j)
19. {
20. i++;
21. }
22. while(arr[j]>mid && i<=j)
23. {
24. j--;
25. }
26. if(i<j)
27. {
28. tmp = arr[i];
29. arr[i] = arr[j];
30. arr[j] = tmp;
31. }
32. }
33. i = 0;
34. mid = arr[(j+1)/2];
35. int b = j + 1;
36. while(i<j)
37. {
38. while(arr[i]<=mid && i<=j)i++;
39. while(arr[j]>mid && i<=j)j--;
40. if(i<j)
41. {
42. tmp = arr[i];
43. arr[i] = arr[j];
44. arr[j] = tmp;
45. }
46. }
47. for(int c = j+1 ;c<b;c++)
48. {
49. cout<<arr[c]<<" ";
50. }
51. cout<<endl;
52. }

### 参考代码二：

void Q\_sort(int\*arr,intl,int r)

{

if(l>=r)

return;

else if( r- l <3)

{

for(int i=l;i<r;i++)

{

for(int k=i+1;k<=r;k++)

{

if(arr[i]>arr[k])

swap(arr[i],arr[k]);

}

}

}

else

{

int mid1 = arr[l],mid2 = arr[(l+r+1)/2],mid3 = arr[r];

int mid = find\_mid(mid1,mid2,mid3);

int key = arr[mid];

while(l<r)

{

while(l<=r &&arr[l]<=key)l++;

while(l<=r && arr[r]>key)r--;

if(l<r)

swap(arr[l],arr[r]);

}

Q\_sort(arr,l,mid-1);

Q\_sort(arr,mid+1,r);

}

}

### F AlvinZH的儿时梦想——木匠篇

### 思路分析

本题。。。。。我只会暴力求解。

首先用结构体储存木棍的x和h,然后用两层循环遍历数组，用long long数据类型比较h\*d\*d的值，比较时可以以（a[i].x\*a[j].x<=0）这个条件剪枝，然后得到最大的h\*d\*d的值后，再强制转换成double类型输出。

然后说说我在这题踩的坑，首先是知道了cin速度比scanf要慢，而且很多题不认取消流同步。然后一开始用的STL中的优先队列，甚至先进行了一次排序，结果依然TLE，得到助教的指导，数据过多时不宜用STL。最后，还是那句话，能用long long就尽量不用int.。

### 算法分析：

暴力循环，复杂度（O(n^2)）

### 参考代码

1. #include<stdio.h>
2. #include<cstdio>
3. #include<cmath>
4. const double pi = acos(-1.0);
5. struct stick
6. {
7. int x;
8. int h;
9. }s[102];
10. int min(int a,int b)
11. {
12. if(a<b)
13. return a;
14. else
15. return b;
16. }
17. int main()
18. {
19. int n,i,j,k;
20. while(scanf("%ld",&n) != EOF)
21. {
22. double V=0.000;
23. for(int k=0;k<n;k++)
24. {
25. scanf("%ld%ld",&s[k].x,&s[k].h);//能用long long 尽量不用int
26. };
27. int d,h;
28. long long sum=0,sum\_t;
29. for(i=0;i<n-1;i++)
30. {
31. for(j=i+1;j<n;j++)
32. {
33. if(s[i].x\*s[j].x<=0)//只靠虑木棒在圆心两边以及在圆心上的情况
34. {
35. d = s[i].x - s[j].x;
36. h = min(s[i].h,s[j].h);
37. sum\_t = h\*d\*d;
38. if(sum\_t> sum)
39. sum = sum\_t;
40. }
41. }
42. }
43. sum = (double)sum;//答案记得要强制转换类型
44. V = sum\*pi/4;
45. printf("%.3lf\n",V);
46. }
47. }

### G D&C--玲珑数

### 思路分析

这道题。。。。以我薄弱的基础，就只能暴力求解了（果断TLE）。在大神的指导下，我参考了归并排序求逆序数问题，顿时领悟到了这道题的精髓。

归并排序的一个好处是，可以在归并的同时完成计数，在本题中也是如此，在归并的时候，由于两边的数据都已经经过排序，所以此时计算玲珑数要快速的多。

注意坑点：数据范围，多次查询，p，q取值。

心得：学到了万能头文件<bits\stdc++.h>

### 算法分析：

归并的最底层划分很大概率上只有两个元素，满足a[i]>2\*a[j]的数，i和j的位置还没有进行改变，这个时候先进行计数，然后往上递归，递归了之后左半部分还是在左边，后半部分还是在右边，再次计数，然后归并，一直归并到最上层，完成计数。

这里的难点在于加法的位置，如果等所有的归并完再相加的话，复杂度还是O（n^2）。

### 参考代码

1. #include<bits/stdc++.h>
2. using namespace std;
3. const int N = 10005;
4. long long arr[N],tmp[N],arr2[N];//多次查询，原数组不变
5. int n,t;
6. int p,q;
7. long long ans;
8. void Merge(int l,int m,int r)
9. {
10. int i=l;
11. int j=m + 1;
12. int k = l;
13. for(;i<=m;i++)
14. {
15. while(j<=r&&arr2[i]>2\*arr2[j]) j++;
16. ans+=j-m-1;
17. }
18. i=l;j=m+1;
19. while(i <= m && j <=r)
20. {
21. if(arr2[i]>arr2[j])
22. {
23. tmp[k++] = arr2[j++];
24. }
25. else
26. {
27. tmp[k++] = arr2[i++];
28. }
29. }
30. while(i<=m) tmp[k++] = arr2[i++];
31. while(j<=r) tmp[k++] = arr2[j++];
32. for(int i=l;i<=r;i++)
33. arr2[i] = tmp[i];
34. }
35. void Merge\_sort(int l,int r)
36. {
37. if(l<r)
38. {
39. int m = (l+r) >> 1;
40. Merge\_sort(l,m);
41. Merge\_sort(m+1,r);
42. Merge(l,m,r);
43. }
44. }
45. int main()
46. {
47. cin>>n;
48. for(int i=0;i<n;i++)
49. {
50. cin>>arr[i];
51. }
52. cin>>t;
53. for(int j=0;j<t;j++)
54. {
55. for(int i=0;i<n;i++)
56. {
57. arr2[i] = arr[i];
58. }
59. cin>>p>>q;
60. ans = 0;
61. if(p>q) swap(p,q);//p,q取值神坑
62. Merge\_sort(p,q);
63. cout<<ans<<endl;
64. }
65. }